The PFLOTRAN Reaction Sandbox

Glenn E. Hammond¹

¹Environmental Subsurface Science Group, Pacific Northwest National Laboratory, Richland, WA 99352

Correspondence: Glenn Hammond (glenn.hammond@pnnl.gov)

Abstract. As modern reactive transport simulators evolve to accommodate the demands of a user community, researchers need a platform for prototyping new biogeochemical processes, many of which are niche and specific to laboratory or field experiments. The PFLOTRAN Reaction Sandbox leverages modern, object oriented Fortran in an attempt to provide such an environment within an existing reactive transport simulator. This work describes the PFLOTRAN Reaction Sandbox concept and implementation through several illustrative examples. Reaction Sandbox Biodegradation Hill customizes the existing microbially-mediated biodegradation reaction formulation within PFLOTRAN to better match empirical data. Reaction Sandbox Simple provides an isolated environment for testing numerous preconfigured kinetic rate expressions and developing user intuition. Reaction Sandbox Example serves as a template for creating new sandboxes within PFLOTRAN.

1 Introduction

Modern reactive transport simulators incorporate sophisticated networks of reactions to simulate complex biogeochemical processes within the Earth’s subsurface environment in support of scientific research in climate change, fate and transport of contaminants, and water resources management (Steefel et al., 2005). As these simulators mature and evolve over time, they can accumulate a large number of chemical reactions with a wide range of implementations. Many reactions are somewhat common among reactive transport codes [e.g., aqueous complexation, mineral precipitation-dissolution, radioactive decay and sorption (Steefel et al., 2015)], while others are niche, prototypical or problem dependent.

There is a multitude of possible reactions to include within a simulator, and the approach to implementing these reactions often varies among simulators. For instance, sorption processes includes absorption, adsorption and ion exchange. Surface complexation is an adsorptive process that can be simulated using a constant capacitance, (diffuse) double layer, triple layer, or non-electrostatic model (Bethke, 2007). Depending on the reaction timescales, surface complexation may be simulated assuming local equilibrium or driven by a kinetic rate expression, and kinetic approaches may include single and multi-rate models, the latter utilizing a distribution of rate constants associated with size fractions of sediment material (Liu et al., 2008).

It is also common for researchers to develop one-off implementations of reactions when existing capabilities cannot replicate biogeochemical phenomenon observed in field or laboratory experiments. For example, Tutolo et al. (2018) demonstrated that the brucite silicification is a serpentinization rate-limiting reaction that is exponentially dependent upon aqueous silica activity. Their brucite silicification reaction required the implementation of a custom rate law that was exponentially dependent upon the activity of aqueous silica. These one-off implementations quickly increase the diversity of reactions supported by a simulator.
As the number of reactions implemented within a simulator grows, and the reactions become increasingly diverse, adding new reactions can challenge ongoing simulator development and maintenance, especially within an open source community where code development is often crowdsourced.

Due to time and funding limitations, it is difficult for code developers to satisfy the needs of the user community by implementing all variants of a reactive process. Nor does it make sense when reactions are problem-specific and may never be used in the future. Meanwhile, the end users who are requesting customization are often non-computational domain scientists with less interest in or limited understanding of numerical methods, programming paradigms, code abstractions, and data layout. Their focus is more on improving predictive accuracy through the refinement of reaction conceptual models, not code development.

This raises the question as to what steps can be taken in the design and implementation of a reactive transport simulator to facilitate code development and long-term maintenance while flattening the (often) steep learning curve for new developers.

The PFLOTRAN Reaction Sandbox is an attempt to provide such an environment within an existing reactive transport simulator. The purpose of the Reaction Sandbox is to provide a means for testing alternative implementations for kinetically formulated rate expressions or networks of these reactions in conjunction with the existing reactive transport capability within PFLOTRAN. Within computer science, the term sandbox often refers to an environment for implementing and vetting new, untested algorithms in isolation. The purpose of a sandbox is to limit the impact on the remainder of the code.

The following sections document the implementation of the PFLOTRAN Reaction Sandbox and demonstrate its application on several problem scenarios. Section 2 provides an overview of PFLOTRAN by presenting the governing equations for reactive transport and the numerical methods employed to solve the resulting discrete nonlinear systems of equations. The section also presents PFLOTRAN’s conventional reactive transport capability and discusses the code’s modular object oriented-design that facilitates the implementation of the Reaction Sandbox. Section 3 describes the foundational (Fortran) Reaction Sandbox class upon which all Reaction Sandbox classes are coded and the high-level programming interface through which the remainder of the code accesses sandbox reactions. Section 4 documents several example sandboxes from which a researcher may derive their own implementation. Finally, Section 5 summarizes the approach.

2 Background

PFLOTRAN is an massively parallel, reactive multiphase flow and transport simulator for modeling subsurface earth system processes (Hammond et al., 2014). The code has been developed under open source GNU LGPL licensing since 2009 and contains contributions from an international group of developers. PFLOTRAN is written in Fortran 2003/2008 which facilitates object oriented design through the use of nested derived types, classes (extensible derived types with member procedures) and procedure pointers. The code is designed as a nested hierarchy of objects. The top-level simulation object contains pointers to all process models, solvers and supporting data (state variables, parameters, etc.) needed to run a simulation.

PFLOTRAN simulates biogeochemical transport through its reactive transport process model. Supported biogeochemical reaction capabilities include aqueous speciation with ion activity models, general Nth-order forward (and reversible) kinetic reactions, microbially-mediated reactions, mineral precipitation-dissolution, radioactive decay and ingrowth and sorption (An-
These reactions are referred to as conventional reactive transport capability as the implementations are based on commonly accepted approaches that are well-documented in the literature.

2.1 Governing Equations

PFLOTRAN’s governing mass conservation equation for reactive transport of aqueous species \( j \) is

\[
\frac{\partial}{\partial t} \left( \phi s \Psi_j \right) + \nabla \cdot \left( q - \phi s D \nabla \right) \Psi_j = Q_j - \sum_r \nu_{jr} I_r,
\]

(1)

with porosity \( \phi \), liquid saturation \( s \), total aqueous component concentration \( \Psi_j \), Darcy fluid flux \( q \), diagonal hydrodynamic dispersion tensor \( D \) and source term \( Q_j \). \( \nu_{jr} \) represents the stoichiometry of species \( j \) in kinetic reaction \( I_r \). Following the continuum formulation for reactive transport (Lichtner, 1985) and assuming local equilibrium (Rubin, 1983), the total aqueous component concentration of species \( j \) is the sum of the free ion concentration \( c_j \) and its stoichiometric contribution \( \nu_{ji} \) to each secondary aqueous complex \( X_i \).

\[
\Psi_j = c_j + \sum_i \nu_{ji} X_i.
\]

(2)

Aqueous complex \( X_i \) is calculated through mass action as

\[
X_i = \frac{K_i}{\gamma_i} \prod_{j'} \left( \gamma_{j'j} c_{j'}^\nu_{j'i} \right)^{\nu_{j'i}},
\]

(3)

with equilibrium constant \( K_i \), activity coefficients \( \gamma_i \) and \( \gamma_{j'i} \), stoichiometry \( \nu_{j'i} \), and primary aqueous species free ion concentration \( c_{j'} \). The governing mass conservation equation for the \( j^{th} \) primary immobile species \( \Phi \) is

\[
\frac{\partial}{\partial t} \left( \Phi_j \right) = -\sum_r \nu_{jr} I_r.
\]

(4)

Ignoring the advection, hydrodynamic dispersion and source terms and assuming constant porosity and saturation, the discrete (finite volume) forms of Eqs. (1) and (4) are

\[
\frac{\phi s V}{\Delta t} \left( \Psi_j^{k+1} - \Psi_j^k \right) \ldots = \ldots - V \sum_r \nu_{jr} I_r
\]

(5)

and

\[
\frac{V}{\Delta t} \left( \Phi_j^{k+1} - \Phi_j^k \right) = -V \sum_r \nu_{jr} I_r,
\]

(6)

respectively. Units for these equations are [mole/second]. Rate expression \( I_r \) represents the primary species mass consumed or produced by each kinetic reaction and has units of [mole/m\(_3\) bulk-second].

2.2 Numerical Solution Technique

PFLOTRAN employs the finite volume method for spatial discretization, backward Euler time integration and Newton’s method for solving the resulting nonlinear system of equations. Newton’s method converges to a solution for the primary
species concentrations $x$ by iteratively evaluating the residual function

$$f(x^p) = 0$$  

(7)

and Jacobian $J$, and solving the linear system

$$J \delta x = -f(x^p)$$

(8)

for the concentration update $\delta x$

$$x^{p+1} = x^p + \delta x.$$  

(9)

$p$ is iteration number.

The residual function $f$ is evaluated by rearranging Eqs. (5) and (6) and setting them equal to zero. For example, the residual for aqueous degree of freedom $c_n$ is

$$f(c^p_n) = \frac{\phi s V}{\Delta t} (\Psi^{k+1}_n - \Psi^k_n) + V \sum_r \nu_{nr} I_r = 0,$$

(10)

while for immobile degree of freedom $\Phi_n$, it is

$$f(\Phi^p_n) = \frac{V}{\Delta t} (\Phi^{k+1}_n - \Phi^k_n) + V \sum_r \nu_{nr} I_r = 0.$$  

(11)

Since the focus of this research is chemical reaction, the advection, dispersion and source terms (represented by ellipses ...) are ignored. The Jacobian contains derivatives of the residual with respect to the primary unknowns, i.e.

$$J_{n,m} = \frac{\partial f(x^p_n)}{\partial x^p_m}.$$  

(12)

Units for internal PFLOTRAN variables are documented in Table 1. These units must be considered in the development of a Reaction Sandbox.

2.3 Object-Oriented Fortran

Modern programming paradigms facilitate the modularity, extensibility and overall longevity of a software product. A common feature among most modern programming paradigms is support for object-oriented design, where objects contain the data structures and procedures necessary to provide functionality, and interfaces are set up for interaction between objects. One major benefit of object oriented design is that when programmed correctly, modifications to an object’s data structures and procedures have little to no impact on other portions of the code. This modularity greatly facilitates the initial development and long-term maintenance of a code.

Modern Fortran facilitates object-oriented design through the use of derived types and classes. A Fortran derived type is a container that encapsulates other Fortran data types (e.g., logicals, integers, reals, other derived types, or pointers to data types). A Fortran class is an extensible version of the derived type that supports inheritance of (type-bound) member variables and
Table 1. Units for PFLOTRAN internal variables.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Symbol</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Porosity</td>
<td>φ</td>
<td>m³ pore / m³ bulk volume</td>
</tr>
<tr>
<td>Liquid saturation</td>
<td>s</td>
<td>m³ liquid / m³ pore</td>
</tr>
<tr>
<td>Volume</td>
<td>V</td>
<td>m³ bulk volume</td>
</tr>
<tr>
<td>Time</td>
<td>t</td>
<td>seconds</td>
</tr>
<tr>
<td>Aqueous free ion concentration</td>
<td>c</td>
<td>mole / kg water (or molality)</td>
</tr>
<tr>
<td>Total aqueous component concentration</td>
<td>Ψ</td>
<td>mole / L water (or molarity)</td>
</tr>
<tr>
<td>Immobile concentration</td>
<td>Φ</td>
<td>mole / m³ bulk volume</td>
</tr>
<tr>
<td>Stoichiometry</td>
<td>ν</td>
<td>-</td>
</tr>
<tr>
<td>Kinetic rate</td>
<td>Iₚ</td>
<td>mole / m³ bulk volume - second</td>
</tr>
<tr>
<td>Residual function</td>
<td>f(xₚ)</td>
<td>mole / second</td>
</tr>
<tr>
<td>Derivative of residual wrt free ion</td>
<td>∂f(xₚ)ₚ</td>
<td>kg water / second</td>
</tr>
<tr>
<td>Derivative of residual wrt immobile</td>
<td>∂f(xₚ)ₚ</td>
<td>m³ bulk volume / second</td>
</tr>
</tbody>
</table>

procedures. A child class, created by extending the parent derived type, inherits all parent class variables and procedures. New member variables may be added to child classes and member procedures may be overridden. By default, a member procedure receives the class object as the first entry in function or subroutine argument lists. (A class object is simply an instantiation of the class.) The procedure uses the object’s member variables and the remaining arguments to perform calculations. Chapman (2018) describes Fortran classes in greater detail.

PFLOTRAN is designed as a nested hierarchy of dynamically allocated objects from the highest-level simulation object down to low level, cell-centric auxiliary objects that store all state variables for each grid cell. Given a pointer to the top-level simulation object, the developer has access to all underlying data structures or objects. This hierarchy engenders modularity and structure within the code. PFLOTRAN employs modern Fortran 2003/2008 where all objects are instantiations of Fortran derived types or classes. It also leverages pointers to procedures and common procedure interfaces to allow users to choose from a suite of constitutive relations, equations of state, flux algorithms, etc., in support of run time options. The modularity afforded through object-oriented Fortran has greatly facilitated the incorporation of new algorithms within PFLOTRAN, as will be shown through the implementation of the Reaction Sandbox.

3 Reaction Sandbox Approach

The PFLOTRAN Reaction Sandbox provides a simplified interface for implementing new kinetically formulated reactions (rate expressions) within PFLOTRAN that are tailored to specific user needs. The term $I_r$ on the right side of Eqs. (1) and (4) represents these kinetic reactions in the governing equations. The Reaction Sandbox isolates one-off or application-specific reactions from the remainder of the PFLOTRAN code base and facilitates code development and long-term maintenance. For the domain scientist desiring to implement a new reaction, the Reaction Sandbox reduces complexity by exposing only the
limited set of variables that are necessary to calculate rate expressions. Thus, the researcher is better shielded from the intricate
details of code development. For kinetic reactions that have the potential for wider acceptance, the Reaction Sandbox provides
a venue for vetting reactions in isolation prior to adoption within the main code base. This section describes the Reaction
Sandbox concept and defines the underlying data structures and user interface.

3.1 Concept

Figure 1 illustrates the hierarchical structure of (hypothetical) Reaction Sandboxes within PFLOTRAN where reaction classes
are derived as descendants of the Reaction Sandbox Base Class. Each Reaction Sandbox class is implemented as a separate
module within the PFLOTRAN source code. The end-user specifies the reaction sandboxes to be employed within the PFLO-
TRAN input file, and a linked list of sandbox reactions is constructed during simulation initialization. Figure 2 illustrates a

Figure 1. Schematic of a hypothetical Reaction Sandbox class hierarchy.

representative linked list of sandboxes composed of two reaction objects from Fig. 1. The outer Reaction Sandbox module
loops over the linked list to perform all operations (e.g., setup, evaluation, destruction, etc.) as illustrated later in Code Block 2.
Linked lists improve flexibility for the code developer as reactions may be inserted or appended in any order. Newly developed
reaction classes are added to the source code as daughter classes in new modules, and instantiated objects are added to the list
during simulation initialization.
3.2 Implementation

The Reaction Sandbox is founded upon two files within the PFLOTRAN source code: reaction_sandbox_base.F90 and reaction_sandbox.F90.

3.2.1 reaction_sandbox_base.F90

reaction_sandbox_base.F90 defines the base (or parent) Fortran class reaction_sandbox_base_type that the developer extends to create new (child) Reaction Sandbox classes. Code Block 1 illustrates the member variables and procedures within the reaction_sandbox_base_type class.


```
type, abstract, public :: reaction_sandbox_base_type
  class(reaction_sandbox_base_type), pointer :: next
contains
  procedure, public :: ReadInput => BaseReadInput
  procedure, public :: Setup => BaseSetup
  procedure, public :: Evaluate => BaseEvaluate
  procedure, public :: UpdateKineticState => BaseUpdateKineticState
  procedure, public :: AuxiliaryPlotVariables => BaseAuxiliaryPlotVariables
  procedure, public :: Destroy => BaseDestroy
end type reaction_sandbox_base_type
```

This class contains a single member variable next, a pointer to the next class of the same reaction_sandbox_base_type type that enables the creation of an abstract linked list of Reaction Sandbox objects. The class also contains empty member procedures with prescribed subroutine interfaces (or argument lists) that the developer overrides in child classes. With the exception of BaseEvaluate, these procedures are empty and return immediately if not overridden by the child class. BaseEvaluate must be extended, and error messaging is incorporated within BaseEvaluate to ensure correct implementation. The other member procedures are optional and do not require implementation in the child classes. Appendix A documents the BaseXXX member procedure interfaces.

3.2.2 reaction_sandbox.F90

reaction_sandbox.F90 serves as the main driver interface for the Reaction Sandbox, providing subroutines that manage the creation, reading, setup, execution and destruction of all Reaction Sandbox objects. For example, the subroutine RSandboxRead instantiates Reaction Sandbox objects based on the keywords parsed from the REACTION_SANDBOX block in the input file. RSandboxEvaluate calculates reaction rates by traversing the linked list of Reaction Sandboxes evaluating individual rates as shown in Code Block 2 (subroutine argument lists have been omitted for simplicity).

```fortran
subroutine RSandboxEvaluate(...) 

  ... 

  class(reaction_sandbox_base_type), pointer :: cur_reaction 

  cur_reaction => rxn_sandbox_list 
  do 
    if (.not.associated(cur_reaction)) exit 
    call cur_reaction%Evaluate(...) 
    cur_reaction => cur_reaction%next 
  enddo 

end subroutine RSandboxEvaluate 
```

With the exception of RSandboxRead, all member procedures are executed from within a linked list loop similar to that shown in Code Block 2. Thus, subroutines within reaction_sandbox.F90 serve as interfaces to the linked lists of Reaction Sandbox objects, and all information is exchanged through these routines. The outer RSandboxXXX subroutines defined in reaction_sandbox.F90 may be called from other PFLOTRAN modules (e.g. see the call to RSandboxEvaluate within subroutine RReaction within reaction.F90), but member procedures within the Reaction Sandbox classes may not be called (to preserve data encapsulation).

The following section describes several example Reaction Sandboxes. These examples are implemented within the PFLOTRAN source code and may serve as templates for future sandboxes.

4 Example Reaction Sandboxes

This section illustrates the implementation of a few Reaction Sandboxes within PFLOTRAN. Reaction Sandbox Biodegradation Hill and Reaction Sandbox Flexible Biodegradation Hill were developed to demonstrate the implementation of a microbially-mediated biodegradation reaction. Reaction Sandbox Simple provides a set of preconfigured reactions that may be uncommented, compiled and run to better understand kinetic rate expressions. Reaction Sandbox Example implements a first-order decay reaction. The comments within reaction_sandbox_example.F90 detail the steps necessary to implement a new Reaction Sandbox class.
4.1 Biodegradation

4.1.1 Biodegradation Conceptual Model

Consider a microbially-mediated biodegradation reaction with biomass growth and decay over time. The reaction could be expressed as

\[ A_{aq} + 0.25B_{aq} \rightarrow 0.33C_{aq} + D_{aq}, \]  

(R1)

with electron donor \( A_{aq} \) and acceptor \( B_{aq} \) and products \( C_{aq} \) and \( D_{aq} \) [M]. The reaction is mediated by the immobile biomass species \( X_{im} \) [mole biomass/m³ bulk] and inhibited above a \( C_{aq} \) concentration of \( 10^{-4} \). The reaction rate \( I_r \) [mole/m³ bulk/sec] can be calculated using Michaelis-Menten kinetics as

\[ I_r = \frac{k_{max}X_{im}A_{aq}}{K_{A_{aq}} + A_{aq}} \times \frac{B_{aq}}{K_{B_{aq}} + B_{aq}} \times \frac{I_{C_{aq}}}{I_{C_{aq}} + C_{aq}} \]  

(13)

with maximum specific utilization rate constant \( k_{max} \) [mole/sec-mole biomass], half saturation constants \( K_{A_{aq}} \) and \( K_{B_{aq}} \) [M], and inhibitor concentration \( I_{C_{aq}} \) [M]. The rate of biomass growth and decay can be modeled as

\[ \frac{dX}{dt} = \text{yield}_{X_{im}} I_r - k_{\text{decay}} X \]  

(14)

with yield \( \text{yield}_{X_{im}} \) [mole biomass/mole] and decay rate constant \( k_{\text{decay}} \) [1/sec]. These rate expressions are implemented as microbial and biomass decay reactions within PFLOTRAN and enabled through the MICROBIAL_REACTION and IMMÔBLE_DECAY_REACTION keywords in the CHEMISTRY block of the input file.

Figure 3 shows PFLOTRAN simulation results for an example batch experiment run over seven days employing the reactions in Eqs. (13) and (14), reaction parameters in Table 2, stoichiometries in Reaction (R1), and initial conditions in Table 3. Results plotted in the figure may be replicated through the following commands:

```
cd $PFLOTRAN_DIR/regression_tests/default/reaction_sandbox
$PFLOTRAN_DIR/src/pflotran/pflotran -input_prefix biodegradation
python biodegradation_vs_data.py biodegradation-obs-0.pft
```

The plot shows the time evolution of aqueous and immobile species concentrations as the week-long simulation runs with a maximum time step size of one hour. The results demonstrate that electron donor \( A_{aq} \) is the limiting substrate, as the concentrations for acceptor \( B_{aq} \) and the reaction products \( C_{aq} \) and \( D_{aq} \) plateau when \( A_{aq} \) is nearly depleted at approximately four days. Biomass concentration increases through 2.25 days at which time the first-order decay rate exceeds the growth rate, and the population begins to fade. Superimposed on the figure are hypothetical experimental results for species \( A_{aq} \) (shown as circles) that deviate from the (blue) simulated curve in the log scale plot. In this example, Eq. (13) is somewhat inaccurate in predicting the tailing behavior of species \( A_{aq} \) at late times.
Figure 3. Time evolution of aqueous and immobile biomass species in a week-long batch biodegradation experiment.

Table 2. Microbially-mediated reaction parameters for the batch biodegradation experiment. $n$ only applies to the reaction incorporating the Hill function (i.e. Eq. (15)). M signifies molarity or mole per liter of water.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_{max}$</td>
<td>9.0e-2</td>
<td>mole/mole biomass-sec</td>
</tr>
<tr>
<td>$K_{A_{aq}}$</td>
<td>2.0e-4</td>
<td>M</td>
</tr>
<tr>
<td>$K_{B_{aq}}$</td>
<td>1.25e-5</td>
<td>M</td>
</tr>
<tr>
<td>$I_{C_{aq}}$</td>
<td>2.5e-4</td>
<td>M</td>
</tr>
<tr>
<td>$yield_{X_{im}}$</td>
<td>1.0e-4</td>
<td>mole biomass/mole</td>
</tr>
<tr>
<td>$k_{decay}$</td>
<td>1.0e-6</td>
<td>1/sec</td>
</tr>
<tr>
<td>$n$</td>
<td>1.2</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3. Initial concentrations for the batch biodegradation experiment.

<table>
<thead>
<tr>
<th>Species</th>
<th>Concentration</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{aq}$</td>
<td>1.0e-3</td>
<td>M</td>
</tr>
<tr>
<td>$B_{aq}$</td>
<td>5.0e-4</td>
<td>M</td>
</tr>
<tr>
<td>$C_{aq}$</td>
<td>1.0e-10</td>
<td>M</td>
</tr>
<tr>
<td>$D_{aq}$</td>
<td>1.0e-10</td>
<td>M</td>
</tr>
<tr>
<td>$X_{im}$</td>
<td>1.0e-4</td>
<td>mole/m^3 bulk volume</td>
</tr>
</tbody>
</table>

The discrepancy in $A_{aq}$ concentration can be resolved by employing a Hill function (Hill, 1910) within the monod expression for $A_{aq}$. Here, the $A_{aq}$ concentration and corresponding half saturation constant are raised to the power $n$.  

$$I_r = k_{max}X_{im} \frac{A_{aq}^n}{K_{A_{aq}} + A_{aq}^n} \times \frac{B_{aq}}{K_{B_{aq}} + B_{aq}} \times \frac{I_{C_{aq}}}{I_{C_{aq}} + C_{aq}}$$  \[(15)\]
However, the Hill function is not an option in PFLOTRAN; the code must be altered to accommodate this new feature. To further complicate matters, the researcher cannot modify the existing Monod expression within PFLOTRAN, since in doing so, the Hill function would be applied to both the $A_{aq}$ and $B_{aq}$ Monod expressions. One possible solution is to implement Eq. (15) as a new reaction, and the Reaction Sandbox is designed to facilitate this process.

4.1.2 Reaction Sandbox Biodegradation Hill

Reaction Sandbox Biodegradation Hill implements the enhanced biodegradation reaction that incorporates the Hill function in Eq. (15) and the biomass growth and decay reaction in Eq. (14). The reactions are encoded in `reaction_sandbox_biohill.F90`. The `reaction_sandbox_biohill_type` class is presented in Code Block 3 where integer IDs for each species are stored as class variables and the Setup and Evaluate procedures are redirected to local BioHillXXX implementations.

**Code Block 3.** Class `reaction_sandbox_biohill_type` in `reaction_sandbox_biohill.F90`.

```fortran
  type, public, &
    extends(reaction_sandbox_base_type) :: reaction_sandbox_biohill_type

! Aqueous species
  PetscInt :: species_Aaq_id
  PetscInt :: species_Baq_id
  PetscInt :: species_Caq_id
  PetscInt :: species_Daq_id

! Immobile species (e.g. biomass)
  PetscInt :: species_Xim_id
contains
  procedure, public :: Setup => BioHillSetup
  procedure, public :: Evaluate => BioHillEvaluate
end type reaction_sandbox_biohill_type
```

Code Block 4 illustrates the assignment of species IDs in an abbreviated version of `BioHillSetup`, based on keywords for aqueous species $A_{aq}$, $B_{aq}$, $C_{aq}$, and $D_{aq}$ and immobile species $X_{im}$ specified in the input file. `BioHillEvaluate` utilizes these IDs to access species concentrations and entries in the residual vector.

**Code Block 4.** Subroutine `BioHillSetup` in `reaction_sandbox_biohill.F90`.

```fortran
  subroutine BioHillSetup(this, reaction, option)
    ...
    ! Aqueous species
    word = 'Aaq'
    this%species_Aaq_id = &
      GetPrimarySpeciesIDFromName(word, reaction, option)
  end subroutine
```
... ! Immobile species
word = 'Xim'
this%species_Xim_id = &
    GetImmobileSpeciesIDFromName(word, reaction%immobile, option)

end subroutine BioHillSetup

Code Block 5 presents an abbreviated version of the implementation of BioHillEvaluate where the naming convention for local variables corresponds closely to reaction parameters and state variables defined in Eqs. (14) and (15). Note that in the calculation of rate \( I_r \), the concentration of \( A_{aq} \) and half saturation constant \( K_{A_{aq}} \) are raised to the power \( n \).

**Code Block 5. Subroutine BioHillEvaluate in reaction_sandbox_biohill.F90.**

```
subroutine BioHillEvaluate(this, Residual, Jacobian, compute_derivative, &

... k_max = 9.d-2
k_decay = 1.d-6
K_Aaq = 2.d-4
K_Baq = 1.25d-5
I_Caq = 2.5d-4

yield = 1.d-4
n = 1.2d0

stoichA = -1.d0
stoichB = -0.25d0
stoichC = 0.33d0
stoichD = 1.d0

I_r = k_max * Xim * Aaq**n / (K_Aaq**n + Aaq**n) * &
    Baq / (K_Baq + Baq) * &
    I_Caq / (I_Caq + Caq)

I = I_r * volume
```
RateA = stoichA * I

... RateX = yield * I - k_decay * Xim * volume

... end subroutine BioHillEvaluate

Figure 4 illustrates a more accurate simulation result for the batch reaction experiment where the Hill function exponent \( n \) is set to 1.2. It is clear that the addition of the Hill function improves the model’s ability to capture the tailing of species \( A_{aq} \) at low concentrations.

![Time History - Linear Scale](image1)

![Time History - Log Scale](image2)

**Figure 4.** Improved match to the batch experiment results for species \( A_{aq} \) after incorporating the Hill function in Eq. (15). Compare with the log-scale plot (without the Hill function) in Fig. 3.

Results plotted in Fig. 4 may be replicated through the following commands:

```bash
cd $PFLOTRAN_DIR/regression_tests/default/reaction_sandbox
$PFLOTRAN_DIR/src/pflotran/pflotran -input_prefix biodegradation_hill
python biodegradation_vs_data.py biodegradation_hill-obs-0.pft
```

The implementation of the enhanced biodegradation reaction in Reaction Sandbox Biodegradation Hill is somewhat rigid. All reaction parameters (rate constants, half saturation constants, stoichiometries, etc.) are hardcoded within the source code as shown in Code Block 5 and may not be changed without code modifications. However, the implementation may be generalized, and this is demonstrated in Reaction Sandbox Flexible Biodegradation Hill.
4.1.3 Reaction Sandbox Flexible Biodegradation Hill

Reaction Sandbox Flexible Biodegradation Hill employs the same biodegradation, growth and decays reactions as Reaction Sandbox Biodegradation Hill with the added flexibility of specifying reaction parameters at run time through the input file, eliminating the need to re-compile PFLOTRAN every time a parameter changes. The new class is implemented in reaction_sandbox_flexbiohill.F90. The reaction_sandbox_flexbiohill_type class extends reaction_sandbox_biohill_type as presented in Code Block 6. The child class inherits the species integer IDs from the parent and adds member variables for storing all reaction parameters and a logical flag for specifying the units of half saturation constants $K_{Aaq}$ and $K_{Baq}$ and inhibitor concentration $I_{Caq}$ (molality versus molarity). The dynamic stoich array enables the use of do loops in the Evaluate routine. The class redirects the ReadInput, Setup, Evaluate and Destroy procedures to local implementations, though only the implementation of FlexBioHillEvaluate will be described below.

**Code Block 6.** reaction_sandbox_flexbiohill_type in reaction_sandbox_flexbiohill.F90.

```fortran
  type, public, &, extends(reaction_sandbox_biohill_type) :: reaction_sandbox_flexbiohill_type
  PetscReal :: k_max
  PetscReal :: K_Aaq_n
  PetscReal :: K_Baq
  PetscReal :: I_Caq
  PetscReal :: yield
  PetscReal :: k_decay
  PetscReal :: n
  PetscBool :: molarity_units
  PetscReal, pointer :: stoich(:)
contains
  procedure, public :: ReadInput => FlexBioHillReadInput
  procedure, public :: Setup => FlexBioHillSetup
  procedure, public :: Evaluate => FlexBioHillEvaluate
  procedure, public :: Destroy => FlexBioHillDestroy
end type reaction_sandbox_flexbiohill_type
```

The FlexBioHillEvaluate routine shown in Code Block 7 is more succinct than BioHillEvaluate (Code Block 5). Reaction parameters are no longer hardwired, but read from the input file in FlexBioHillReadInput and stored as class member variables. The use of the class member stoich array within the do loops eliminates the need to hardwire the Residual array indexing and reduces the number of lines of code. In addition, FlexBioHillEvaluate demonstrates the ability to choose between molality versus molarity for aqueous concentrations and half saturation constants, a useful option.
since laboratory data is often available in either format. There is also support for calculating analytical derivatives (for the Jacobian), which can be programmed much more concisely with the stoich array and do loops (not shown).
observation at a mid-column observation point (x=49.5 m). Table 4 summarizes simulation parameters, while Table 5 describes the initial and boundary concentrations.

**Table 4.** Reaction Sandbox Simple conceptual model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column length</td>
<td>100 m</td>
</tr>
<tr>
<td>Cross-sectional area</td>
<td>1 m²</td>
</tr>
<tr>
<td>Grid resolution</td>
<td>1 m</td>
</tr>
<tr>
<td>Prescribed Darcy velocity</td>
<td>1 m/y</td>
</tr>
<tr>
<td>Pore water velocity</td>
<td>4 m/y</td>
</tr>
<tr>
<td>Porosity</td>
<td>0.25</td>
</tr>
<tr>
<td>Tortuosity</td>
<td>1.0</td>
</tr>
<tr>
<td>Water density</td>
<td>1000 kg/m³</td>
</tr>
<tr>
<td>Aqueous diffusion coefficient</td>
<td>$10^{-9}$ m²</td>
</tr>
<tr>
<td>Observation point location</td>
<td>49.5 m</td>
</tr>
<tr>
<td>Initial timestep size</td>
<td>1 h</td>
</tr>
<tr>
<td>Maximum timestep size</td>
<td>0.25 y</td>
</tr>
<tr>
<td>Final simulation time</td>
<td>25 y</td>
</tr>
<tr>
<td>Observation output frequency</td>
<td>Every timestep</td>
</tr>
<tr>
<td>Snapshot file output times</td>
<td>0, 6.25, 12.5, 18.75, 25 y</td>
</tr>
</tbody>
</table>

**Table 5.** Initial and boundary concentrations in the Reaction Sandbox Simple input file.

<table>
<thead>
<tr>
<th>Species</th>
<th>Initial Concentration</th>
<th>Boundary Concentration</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{aq}$</td>
<td>1.e-10</td>
<td>1.e-3</td>
<td>M</td>
</tr>
<tr>
<td>$B_{aq}$</td>
<td>1.e-10</td>
<td>1.e-3</td>
<td>M</td>
</tr>
<tr>
<td>$C_{aq}$</td>
<td>1.e-10</td>
<td>1.e-10</td>
<td>M</td>
</tr>
<tr>
<td>$D_{aq}$</td>
<td>1.e-10</td>
<td>1.e-10</td>
<td>M</td>
</tr>
<tr>
<td>$E_{aq}$</td>
<td>1.e-10</td>
<td>1.e-10</td>
<td>M</td>
</tr>
<tr>
<td>$F_{aq}$</td>
<td>1.e-10</td>
<td>1.e-10</td>
<td>M</td>
</tr>
<tr>
<td>$X_{im}$</td>
<td>1.e-4</td>
<td>N/A</td>
<td>mole/m³ bulk volume</td>
</tr>
<tr>
<td>$Y_{im}$</td>
<td>1.e-10</td>
<td>N/A</td>
<td>mole/m³ bulk volume</td>
</tr>
</tbody>
</table>

The implementation of the `reaction_sandbox_simple_type` class is nearly identical to that of `reaction_sandbox_biohill_type` in Code Block 3 with the addition of member variables `species_Eaq_id`, `species_Faq_id` and `species_Yim_id`. Member procedure `SimpleSetup` links these integer IDs to their respective names from the input file. The user may evaluate reactions in this sandbox with the following steps:
1. Uncomment a rate expression block within subroutine SimpleEvaluate.

```
cd $PFLOTRAN_DIR/src/pflotran
[emacs, gedit, nano, vi] reaction_sandbox_simple.F90

Remove "!uncomment:" prefixes from lines within chosen rate expression in subroutine SimpleEvaluate.
Save the file.
```

2. Compile the PFLOTRAN executable.

```
make pflotran
```

3. Navigate to $PFLOTRAN_DIR/regression_tests/default/reaction_sandbox

```
cd $PFLOTRAN_DIR/regression_tests/default/reaction_sandbox
```

4. Change the x-direction grid resolution in reaction_sandbox_simple.in to 100.

```
[emacs, gedit, nano, vi] reaction_sandbox_simple.in

NXYZ 10 1 1 → NXYZ 100 1 1
```
Save the file.

5. Run the simulation.

```
$PFLOTRAN_DIR/src/pflotran/pflotran -input_prefix reaction_sandbox_simple
```

6. Plot the results with reaction_sandbox_simple.py.

```
python reaction_sandbox_simple.py
```

Code Block 8 illustrates a commented rate expression block for calculating the first order decay of species Aaq to daughter product Caq.

---

**Code Block 8. Commented first-order rate expression block in SimpleEvaluate within reaction_sandbox_simple.F90.**

```
subroutine SimpleEvaluate(...)

... 

! first-order (A -> C)
!uncomment: k = 1.d-9 ! [1/sec]
!uncomment: stoichA = -1.d0
!uncomment: stoichC = 1.d0
!uncomment: Rate = k * Aaq * L_water ! [mol/sec]
!uncomment: RateA = stoichA * Rate
```
---
The user deletes all “!uncomment:” prefixes in the codeblock as shown in Code Block 9, compiles PFOTRANS, navigates to the reaction_sandbox folder, and runs the code.

**Code Block 9.** Uncommented first-order rate expression block in SimpleEvaluate within reaction_sandbox_simple.F90.

```fortran
subroutine SimpleEvaluate(...)
  ...
  ! first-order (A → C)
  k = 1.d-9 ! [1/sec]
  stoichA = -1.d0
  stoichC = 1.d0
  Rate = k * Aaq * L_water ! [mol/sec]
  RateA = stoichA * Rate
  RateC = stoichC * Rate
  ...
end subroutine SimpleEvaluate
```

Figure 5 illustrates the simulation results. Plotted to the left is concentration breakthrough at the observation point, and to

**Figure 5.** Results from Reaction Sandbox Simple with the first-order A → C rate expression uncommented.
the right is a snapshot of concentration profiles at 12.5 years. Species $A_{aq}$ clearly decays into $C_{aq}$ while the other species are transported without reaction. The user may evaluate the other rate expressions in the subroutine with the same approach.

### 4.3 Reaction Sandbox Example

PFLOTRAN’s `reaction_sandbox_example.F90` serves as a template for implementing new Reaction Sandboxes. The sandbox implements the first-order decay of $A_{aq}$ without daughter products, and the developer modifies the templated source code to incorporate new reactions. Comments within the source code enumerate steps for implementing new reactions beginning with the renaming of subroutines and variables and ending with the implementation of subroutine `ExampleDestroy` at the bottom of the file. In between, source code is modified to implement the new reaction(s).

Code Block 10 illustrates the first two steps embedded within comments in the source code near the top of the file.

**Code Block 10.** Source code with embedded comments from the top of `reaction_sandbox_example.F90`.

```fortran
module Reaction_Sandbox_Example_class

#include "petsc/finclude/petscsys.h"
use petscsys

! 1. Change all references to "Example" as desired to rename the module and
!    subroutines within the module.

use Reaction_Sandbox_Base_class

use Global_Aux_module
use Reactive_Transport_Aux_module
use PFLOTRAN_Constants_module

implicit none

private

! 2. Add module variables here. Note that one must use the PETSc data types
!    PetscInt, PetscReal, PetscBool to declare variables of type integer
!    float/real*8, and logical respectively. E.g.,
!
```
Comments within ExampleEvaluate provide a detailed description of the subroutine’s arguments and local variables, including members of the reaction and reactive transport auxiliary variable classes (i.e., reaction_rt_type and reactive_transport_auxvar_type, respectively) which may be used in rate expression calculations. The units of all variables and the Residual and Jacobian arrays are also provided. Code Block 11 shows several of these detailed comment blocks.

**Code Block 11.** Representative comments from subroutine ExampleEvaluate.

```fortran
! rt_auxvar - Object holding chemistry information (e.g., concentrations, 
!   activity coefficients, mineral volume fractions, etc.). See 
! reactive_transport_aux.F90.
!
! Useful variables:
! rt_auxvar%total(:,iphase) - total component concentrations 
! [mol/L water] for phase
!
! rt_auxvar%pri_molal(:) - free ion concentrations [mol/kg water]
!
! rt_auxvar%pri_act_coef(:) - activity coefficients for primary species
!
! rt_auxvar%aqueous%dtotal(:,iphase) - derivative of total component 
! concentration with respect to free ion [kg water/L water]
...

! 10. Add code for the Residual evaluation.

! Units of the Residual must be in moles/second.
! 1.d3 converts m^3 water -> L water
L_water = material_auxvar%porosity*global_auxvar%sat(iphase)* &
  material_auxvar%volume*1.d3
!
! Always "subtract" the contribution from the Residual.
Residual(this%species_id) = Residual(this%species_id) - &
  (-1.d0) * & ! negative stoichiometry
  this%rate_constant * & ! 1/sec
  L_water * & ! L water
  rt_auxvar%total(this%species_id,iphase) ! mol/L water
...
```
Once refactoring is complete, the developer must modify the corresponding class and subroutine names within reaction_sandbox.F90 to match those in the newly refactored reaction_sandbox_example.F90. The source code may then be compiled without changing the reaction_sandbox_example.F90 filename. Should this filename be revised, the developer must update the corresponding filenames within the pflotran_object_files.txt and pflotran_dependencies.txt files referenced by the makefile.

The best approach to compiling the code is to perform a make clean to remove all previously built modules and object files and the pflotran executable, and then, make pflotran to compile the code. reaction_sandbox_example.in, located in $PFLOTRAN_DIR/regression_tests/default/reaction_sandbox, provides a representative input deck for this example. Note that the card EXAMPLE in the REACTION_SANDBOX block of this input file must be updated to match the corresponding keyword added to reaction_sandbox.F90.

5 Conclusions

Customization of biogeochemical reaction networks is often necessary in the development of reactive transport simulators employed to simulate problem-specific scenarios in the real world. For researchers in the natural sciences, who may be more focused on biology, chemistry and/or physics than computational science, modifying these codes to incorporate new scientific processes can be challenging. The PFLOTRAN Reaction Sandbox may help remedy this issue.

The Reaction Sandbox provides a modular environment for prototyping new kinetic rate expressions that do not exist within PFLOTRAN. Within the Reaction Sandbox, novel reaction networks can evolve and mature over time; natural selection can run its course. Once vetted, these reactions may be incorporated more efficiently elsewhere within the permanent code base.

This work demonstrates the implementation of reactions within the Reaction Sandbox. Several new Reaction Sandbox classes are conceptualized and implemented based on existing rate expressions within PFLOTRAN for microbially-mediated biodegradation to improve the code’s ability to match hypothetical empirical data and provide greater flexibility from the end user perspective. Reaction Sandbox Simple is presented as a means of prototyping numerous common kinetic rate expressions within a preconfigured column experiment context. Reaction Sandbox Example provides a template for implementing new reactions within PFLOTRAN.

Code availability. The source code, input files and results presented in this manuscript are based on PFLOTRAN v4.0. A snapshot of this release is available at https://zenodo.org/record/5826289. The corresponding version of PETSc is v3.16.2 and was configured on Ubuntu 18.04 with GCC 7.5 using the following config script: ./configure -CFLAGS=-O3 -CXXFLAGS=-O3 -FFLAGS=-O3 -with-debugging=no -download-mpich=yes -download-hdf5=yes -download-hdf5-fortran-bindings=yes -download -download-metis=yes -download-parmetis=yes.
Appendix A: Reaction Sandbox Member Procedure Interfaces

This section documents Fortran class reaction_sandbox_base_type member procedure interfaces not covered in Section 3.2.1 (see reaction_sandbox_base.F90).

A1 ReadInput => BaseReadInput

ReadInput provides a customizable interface for reading parameters associated with the reaction sandbox from a block in the input file. The input block is opened by a unique keyword associated with the child Reaction Sandbox class.

Code Block 12. BaseReadInput argument list.

```fortran
subroutine BaseReadInput(this,input,option)
  ...
  class(reaction_sandbox_base_type) :: this
  type(input_type), pointer :: input
  type(option_type) :: option
```

this: reaction_sandbox_base_type object

input: object storing the input file object pointer and line buffers

option: object storing run time options including process rank and an error messaging buffer

A2 Setup => BaseSetup

Setup initializes the Reaction Sandbox class by allocating dynamic memory, mapping species IDs, assigning stoichiometries and rate constants, etc. The degree to which Reaction Sandbox class settings are customizable at run time is up to the developer and their creativity in implementing ReadInput and Setup.


```fortran
subroutine BaseSetup(this,reaction,option)
  ...
  class(reaction_sandbox_base_type) :: this
  class(reaction_rt_type) :: reaction
  type(option_type) :: option
```

this: reaction_sandbox_base_type object

reaction: object storing chemical species and general reaction information

option: object storing run time options including process rank and an error messaging buffer
A3 Evaluate => BaseEvaluate

Evaluate calculates kinetic rates for all reactions in the Reaction Sandbox class and adds the (kinetic rate) contributions to the Residual and Jacobian arrays. This is the only subroutine that must be extended in child Reaction Sandbox classes.


```fortran
subroutine BaseEvaluate(this, Residual, Jacobian, compute_derivative, &
    rt_auxvar, global_auxvar, material_auxvar, &
    reaction, option)
...
class(reaction_sandbox_base_type) :: this
class(reaction_rt_type) :: reaction
! the following arrays must be declared after reaction
PetscReal :: Residual(reaction%ncomp)
PetscReal :: Jacobian(reaction%ncomp, reaction%ncomp)
PetscBool :: compute_derivative
    type(reactive_transport_auxvar_type) :: rt_auxvar
    type(global_auxvar_type) :: global_auxvar
    class(material_auxvar_type) :: material_auxvar
    type(option_type) :: option
```

this: reaction_sandbox_base_type object

Residual: 1D array of double precision numbers holding contributions to the residual equations at each grid cell

Jacobian: 2D array of double precision numbers holding contributions to the Jacobian matrix at each grid cell

compute_derivative: flag toggling on the calculation of analytical derivatives for the Jacobian matrix when true

rt_auxvar: object storing reactive transport state variables (e.g., concentrations, rates, etc.) at each grid cell

global_auxvar: object storing flow state variables (e.g., density, saturation, etc.) at each grid cell

material_auxvar: object storing material and cell properties (e.g., porosity, volume, etc.) at each grid cell

reaction: object storing chemical species and general reaction information

option: object storing run time options including process rank and an error messaging buffer
**A4 UpdateKineticState => BaseUpdateKineticState**

UpdateKineticState updates state variables associated with the Reaction Sandbox class that are stored in rt_auxvar and updated at the end of a time step based on rates calculated in the Reaction Sandbox (e.g., for mass balance calculations, updates to mineral volume fractions, etc.).

**Code Block 15. BaseUpdateKineticState argument list.**

```fortran
subroutine BaseUpdateKineticState(this, rt_auxvar, global_auxvar, &
  material_auxvar, reaction, option)
...
class(reaction_sandbox_base_type) :: this
type(reactive_transport_auxvar_type) :: rt_auxvar
type(global_auxvar_type) :: global_auxvar
class(material_auxvar_type) :: material_auxvar
class(reaction_rt_type) :: reaction
type(option_type) :: option
```

- this: reaction_sandbox_base_type object
- rt_auxvar: object storing reactive transport state variables (e.g., concentrations, rates, etc.) at each grid cell
- global_auxvar: object storing flow state variables (e.g., density, saturation, etc.) at each grid cell
- material_auxvar: object storing material and cell properties (e.g., porosity, volume, etc.) at each grid cell
- reaction: object storing chemical species and general reaction information
- option: object storing run time options including process rank and an error messaging buffer

**A5 AuxiliaryPlotVariables => BaseAuxiliaryPlotVariables**

AuxiliaryPlotVariables appends Reaction Sandbox-specific state variables stored in rt_auxvar to the list of output variables to be printed to observation and snapshot files.

**Code Block 16. BaseAuxiliaryPlotVariables argument list.**

```fortran
subroutine BaseAuxiliaryPlotVariables(this, list, reaction, option)
...
class(reaction_sandbox_base_type) :: this
type(output_variable_list_type), pointer :: list
class(reaction_rt_type) :: reaction
type(option_type) :: option
```
A6 Destroy => BaseDestroy

Destroy deallocates all dynamic memory in the Reaction Sandbox class at the end of a simulation.

```
subroutine BaseDestroy(this)
  ...
  class(reaction_sandbox_base_type) :: this
```

Author contributions. GH implemented the Reaction Sandbox in PFLOTRAN, designed and implemented the example sandboxes, and prepared the manuscript.

Competing interests. The author declares that he has no conflicts of interest.

Acknowledgements. This research was supported by the U.S. Department of Energy (DOE), Office of Biological and Environmental Research (BER), as part of BER’s Environmental System Science Program (ESS). This contribution originates from the River Corridor Scientific Focus Area (SFA) at the Pacific Northwest National Laboratory (PNNL) and was supported by the partnership with the IDEAS-Watersheds. PNNL is operated for the DOE by Battelle Memorial Institute under contract DE-AC05-76RL01830. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.
References


